# HP-CERTI: Towards a high Performance, high Availability Open Source RTI for Composable Simulations (04F-SIW-014)

*Martin Adelantado*[*], *Jean-Loup Bussenot, Jean-Yves Rousselot, Pierre Siron, Marc Betoule*[*]

ONERA-CT
French Aerospace Research Agency
Information Modeling and processing Department (DTIM)
[*]Long Term Design and System Integration Department (DPRS)
2, Avenue Edouard Belin
31055 Toulouse Cedex
France

Phone: +33 (0) 5 62 25 26 61
Fax: +33 (0) 5 62 25 25 93
Email: Adelantado@cert.fr

Keywords:
Composable simulations, High Level Architecture, RTI, SSI operating systems

**ABSTRACT:** *Composing simulations of complex systems from already existing simulation components remains a challenging issue. Motivations for composable simulation include generation of a given federation driven by operational requirements provided "on the fly". The High Level Architecture, initially developed for designing fully distributed simulations, can be considered as an interoperability standard for composing simulations from existing components. Requirements for constructing such complex simulations are quite different from those discussed for distributed simulations. Although interoperability and reusability remain essential, both high performance and availability have also to be considered to fulfill the requirements of the end user. ONERA is currently designing a High Performance / High Availability HLA Run-time Infrastructure from its open source implementation of HLA 1.3 specifications. HP-CERTI is a software package including two main components: the first one, SHM-CERTI, provides an optimized version of CERTI based on a shared memory communication scheme; the second one, Kerrighed-CERTI, allows the deployment of CERTI through the control of the Kerrighed Single System Image operating system for clusters, currently designed by IRISA. This paper describes the design of both high performance and availability Run-time Infrastructures, focusing on the architecture of SHM-CERTI. This work is carried out in the context of the COCA (High Performance Distributed Simulation and Models Reuse) Project, sponsored by the DGA/STTC (Délégation Générale pour l'Armement/Service des Stratégies Techniques et des Technologies Communes) of the French Ministry of Defense.*

## 1. Introduction

While HLA initially began as a freely downloadable software in an attempt to promote a new distributed simulation technology, it has subsequently been commercialized by several companies. Recently, the distributed simulation community claims that developing an open-source RTI would be essential to promote adoption of HLA outside of the current

defense oriented user-base, and to achieve an increased responsiveness to user requirements [3].

Among already existing non-commercial RTIs, CERTI is a C++ implementation of the HLA 1.3 specification designed initially for internal use at ONERA (French Aerospace Research Agency), and has subsequently been made available as an open-source RTI to the simulation community (GPL and LGPL licenses). Current CERTI software architecture is based on three levels of communicating processes. The first process is the RTIG, a global centralized manager transferring messages between several RTIA (RTI Ambassador) processes over the network. The second level is provided by the RTIA which is run locally to each federate, and communicates over TCP or UDP with the RTIG. Finally, the third level comprises the federate code which is exposed to the HLA interface through the libRTI. Communications between libRTI and RTIA are supported by Unix sockets.

While HLA was initially designed to support fully distributed simulation applications, it provides a promising framework for composing not necessarily distributed simulations, from existing reusable components.

Simulation composability allows then to construct federations from a set of communicating components according to the needs of the decision makers. Composability provides a means to construct integrated simulation platforms with increased coverage of decision support. In such simulation applications, distribution becomes a means to achieve high performance computing, while remaining a constraint since existing components are reused.

To face the aforementioned performance and availability requirements, ONERA is currently designing the HP-CERTI package, an optimized version of CERTI, including two main development issues. The first one, named SHM-CERTI, deals with a shared memory communication scheme between RTIG and RTIAs, in order to achieve high performance simulation of federations running on the same shared memory execution platform. The objective of the second issue is to increase both availability and performance of composable simulations running on high performance cluster platforms. To achieve these objectives, CERTI is deployed through the control of a SSI (Single System Image) operating system, named Kerrighed and designed by IRISA (Information Technology and Random Systems Research Institute), Rennes, France. Kerrighed takes benefit of the underlying hardware performance by providing global management of all cluster resources including processor, memory and disk. Moreover, its dynamic resource management policies make cluster configuration changes transparent to the applications and guarantee system availability despite node failures. Since the standard host operating system interface is kept, applications can be executed on Kerrighed without being modified. Kerrighed is implemented as a set of Linux modules and is downloadable under an Open Source license (GPL).

In this paper, we first discuss motivations for composable simulations and both high performance and availability requirements. The next section briefly provides essential background on both CERTI and Kerrighed operating system. In the fourth section, we describe the communication architecture of SHM-CERTI focusing on the underlying communication protocols between RTIG - RTIAs. Section 5 explains how Kerrighed-CERTI is deployed over the Kerrighed SSI operating system. In section 6, we discuss preliminary comparative performance results using a four processor shared memory computer and a cluster of eight biprocessor nodes connected through a high speed Myrinet network, under Linux Red Hat operating system. Finally, conclusion and further works are discussed.

## 2. Motivations for high performance, high availability composable simulations

Modeling and fast-time simulation (FTS) are nowadays very commonly used in several industrial fields such as energy, factories, economic or transportation systems. This is a consequence of the critical need to know in an early stage of development, the answers to "what if?" questions. For example, In the air/ground traffic management system domain, stakeholders and policy makers involved in or affected by the decision making process are asked to make decisions , draw policy directions and operate in a quite complicated environmental, institutional and operational setting. Therefore, policy makers, airport authorities, air traffic services providers, terminal operators and other stakeholding groups that are involved in ground/air operations, often face challenging decision making problems with strong interdependencies and often conflicting objectives. These problems have been addressed both by individual actors and major industry actors such as the European Commission, Eurocontrol or the FAA.

In response to these decision making requirements, new integrated M&S tools infrastructures are necessary. They have to be able to support both high level political decision (strategic level decision making) related to airside, landside, and airspace planning, as well as operations (tactical/operational level decision making) with respect to a variety of

measures of traffic effectiveness (capacity, delay, quality of service, environmental issues, safety). Requirements for such kind of decision making tools include reusability of existing models and tools, interoperability between components and composability allowing to rapidly compose simulations and simulation environments both statically (design time) and dynamically (run-time). In [12], composability has been defined as "the capability to select and assemble components in various combinations into complete, validated simulation environments to satisfy specific user requirements across a variety of application domains, levels of resolution and time scales".

Although HLA has been proposed to facilitate reuse and interoperability of existing federates, distribution of elementary components, potentially worldwide, constitutes a dramatic drawback regarding execution performances of large scale simulations communicating through wide area networks. Typically, high performance is needed to face the requirements of the operational decision making process. For example, slot management or aircraft departure sequence optimization are well known use cases that would be assisted by real time simulation tools.

Our vision of dynamically composable simulations driven by the end user requirements, is based on the assumption that HLA can probably be used as an interoperability standard, and then facilitates reuse of available components devoid of proprietary constraints [5], [6], [7]. Nevertheless, such a vision of operational integrated platforms for constructing decision support systems based on M&S through the HLA middleware requires to significantly increase the execution performances of HLA federations, focusing on the communication performances of the associated RTI.

## 3. Technical Background

### 3.1. CERTI Architecture

CERTI is an open source implementation of the HLA 1.3 specifications, freely downloadable from http://www.cert.fr/CERTI web site.

Motivations for designing CERTI within our French governmental agency, have already been explained elsewhere [1], [2]. In this section, we focus on the main architecture of the Run-time Infrastructure built around a set of communicating processes, and depicted in figure 1. Additional information describing the communication protocol between processes, will be provided in section four. CERTI is a distributed system written in C++ language, involving three main components: a local process RTIA (RTIAmbassador), a centralized process RTIG (RTIGateway) and the libRTI library which has to be linked with each participating federate.

Each federate process locally interacts with its local RTIA through Unix-domain sockets. The RTIA process exchanges messages over the network with the RTIG process, through TCP and UDP sockets, allowing then various distributed algorithms needed by the RTI services, to be run.

The main function devoted to the RTIG is to manage the communications between the RTIAs and consequently between federates. Communications between two federates are supported by two TCP connections: one connection is established between the requesting RTIA and the RTIG, while the second one involves the RTIG and the receiving RTIA. CERTI follows then a centralized architectural approach, in order to significantly simplify the underlying implementation of several HLA services.

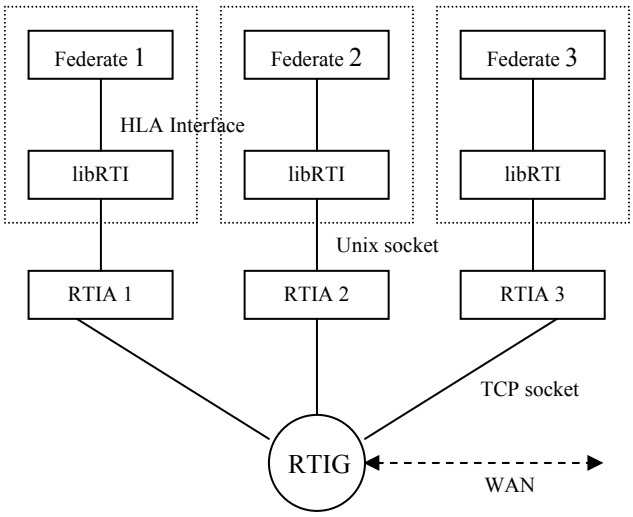


Figure 1: CERTI architecture

The libRTI library contains the HLA services calls which are packaged as messages to be sent to the RTIA. Each service execution waits then for a response from the RTIA process, providing the output parameters. The conventional *tick* primitive has been included allowing then the execution of the RTI initiated services. Finally, the CPU resource allocation policy between both federate and associated RTIA, is managed by the target operating system.

### 3.2. Kerrighed Single System Image Operating System

Clusters are now accepted as an alternative to parallel architectures for high performance computing.

Moreover, several applications that were traditionally executed on multiprocessor machines are now executed on clusters. In the area of scientific computing, several parallel programming models have been proposed, while parallel applications use either multithreading or message-passing libraries such as MPI or PVM. Executing parallel applications on a cluster is not easy, because there is no obvious solution to support multithreaded execution when memory is distributed on all nodes. Secondly, high performance depends on efficient placement and load-balancing strategies to take full advantage of all the resources of the cluster.

To face these requirements, Kerrighed has been designed by IRISA, as a single system image (SSI) operating system for a cluster [8], [9], [10], [11]. Such a system should offer the same interface as a traditional operating system for a SMP machine to designers. Kerrighed is built as an extension to the Linux operating system and is independent from the cluster interconnection network. For more details, please refer to the www.kerrighed.org web site.

# 4. SHM-CERTI Communication Architecture

In this section, we will focus on the communication architecture of the SHM-CERTI, since the software architecture of the RTI itself remains unchanged. Both SHM-CERTI and CERTI are written in C++ language. Notice that from a software development point of view, a single open source package has to be designed, in order to facilitate the development of further releases.

## 4.1. Main communication mechanisms

In the SHM-CERTI architecture, TCP socket based communications between the RTIG and RTIAs are replaced by shared memory segments, which are created by the RTIG process and attached by both RTIG and requesting RTIA. In order to simplify the communication protocol during the simulation main loop, joining and resigning a federation involve two Unix messages files, shared by the RTIG and all the RTIAs. The first one, named RTIAs_to_RTIG is used by each RTIA when it requests a joining operation. The requesting federate waits then the shared memory ID from the RTIG, through the second messages file, named RTIG_to_RTIAs. Once segment accessing and attaching operations have been successfully performed by both clients, data exchanges between RTIG and the incoming RTIA, follow a shared memory based communication scheme.

Conversely, when a given RTIA is intended to resign a federation, the RTIG "invalidates" the corresponding shared memory segment, without destroying it. More precisely, the RTIG maintains a list of all already accessed and attached shared memory segments that can be then reused when a federate joins a federation. Invalidation of shared memory segments avoids accessing and attaching unnecessary operations during the simulation execution.

## 4.2. Data structures and messages format

The main data structures and format of messages exchanged by the CERTI components have remained unchanged in order to reduce the cost of the migration process. Typically, each HLA service call involves a corresponding message based on two components, the header part and the body part. The former identifies the service call, while the latter encodes the corresponding parameters.

To ensure communications through attached shared memory segments, the following structured type has been defined:

```
typedef struct {
        SharedMemory RTIG_to_RTIA;
        SharedMemory RTIA_to_RTIG;
} SHM;
```

The *SharedMemory* type is defined as follows:

```
typedef struct {
        char status;
        HeaderStruct Header;
        MessageBody Body;
} SharedMemory;
```

*HeaderStruct*, respectively *MessageBody*, types obviously corresponds to the header part, respectively the body part of the basic messages.

When an incoming RTIA requests a joining operation, the RTIG attach a given shared memory segment at a pointer to a *SHM* type variable. Therefore, the *RTIG_to_RTIA* buffer is used to support communications from the RTIG to the RTIA, whilst the *RTIA_to_RTIG* buffer ensures data exchanges from the RTIA to the RTIG. The *status* flag of the *SharedMemory* structure indicates the current state (empty/full) of the corresponding shared memory. The synchronisation protocol between both users of the shared memory (RTIA and RTIG) is based on an adequate management of the *status* flag, ensuring that when the $U_1$ user writes a message in the $U_1$_to_$U_2$ buffer, subsequent write operations are blocked until the message has been read by the $U_2$ user.

### 4.3. The simulation main loops

This subsection addresses the main difficulties that have been encountered to migrate the CERTI to the SHM-CERTI. A better understanding of the main simulation loops of both RTIA and RTIG is then necessary. In this subsection, we focus on the RTIG, the RTIA main loop being quite similar. A simplified algorithm of the CERTI RTIG main simulation loop is the following:

*While (!END_SIMULATION)*
*Wait for an incoming message*
*from a given RTIA (**select**)*
*Read the incoming message **(read)***
*Process the message*
*Broadcast the response to the corresponding RTIAs (**write**)*
*Endwhile*

The *select* method is used to select a given communication request or to establish a new connection with a joining federate. Methods *read* and *write* allow to read (respectively write) the header and the body of every message corresponding to a given HLA service. In most cases, processing a given message leads then the RTIG to send several messages to the participating RTIAs. The simplified algorithm of the SHM-CERTI main loop becomes the following:

*Attach both RTIAs_to_RTIG and RTIG_to_RTIAs messages files*

*Blocked wait for the first joining RTIA*

*While (!END_SIMULATION)*
*Check a new RTIA is joining or resigning*
*If (an RTIA is joining)*
*Find a free shared memory segment*
*Send the corresponding ID*
*Else*
*If (an RTIA is resigning)*
*Invalidate the corresponding segment*
*Endif*
*Endif*

*Unblocked wait for a message in a given RTIA_to_RTIG shared memory segment*
*If (a message is available)*
*Read the incoming message **(read)***
*Process the message*
*Broadcast the response to the corresponding RTIAs (**write**)*
*Endif*
*Endwhile*

The *Unblocked wait for a message* method checks if there is an RTIA_to_RTIG shared buffer with the status flag set to full, indicating that an incoming message is available. The *read* and *write* methods are overloaded methods performing the synchronization protocol briefly discussed in section 4.2.

When the *read* method successfully read an available message, the *status* flag is set to empty. In a same way, when the *write* method successfully writes a message, the *status* flag is set to full.

Two versions of both *read* and *write* methods are available. The *blocked read*, respectively *blocked write*, method waits for an incoming message, respectively waits for the corresponding shared memory buffer becomes empty, and reads, respectively writes, the message. Conversely, the *unblocked read* returns the message if it is available and the value 0 otherwise. The *unblocked write* writes the message if the shared memory buffer is empty and returns 0 otherwise.

### 4.4. Communication protocol example

In this subsection, an example of the shared memory communication protocol is proposed. This example is based on the exchange of messages involved in the *UpdateAttributesValue* RTIAmbassador service and *ReflectAttributesValue* FEDAmbassador callback. Notice that message delivery conditions to the federate, are not represented, since they strongly depend on time management. Finally, in figure 2, depicting the communication protocol, the UAV acronym is used for *UpdateAttributesValue*, respectively the RAV acronym for *ReflectAttributesValue*.

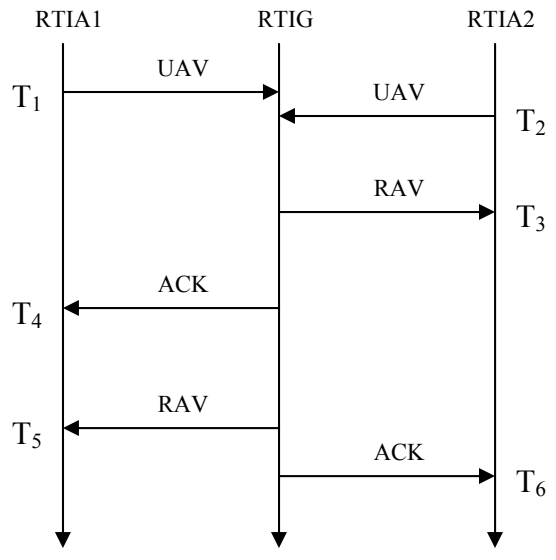


Figure 2: Example of communication protocol

This communication protocol involves two RTIAs invoking concurrent UAV services to the RTIG. In the following, we discuss the corresponding processing steps from both RTIG and RTIAs side, showing then that the communication protocol is deadlock free. For a better understanding of the explanation, remember that two shared memory segments, SEG1 and SEG2, are involved. SEG1 is attached by both RTIG and RTIA1 to the memory buffer SHM1, while SEG2 is attached by both RTIG and RTIA2 to the memory buffer SHM2 (Figure 3).

Each memory buffer is a structure composed of two items, RTIA_to_RTIG and RTIG_to_RTIA.

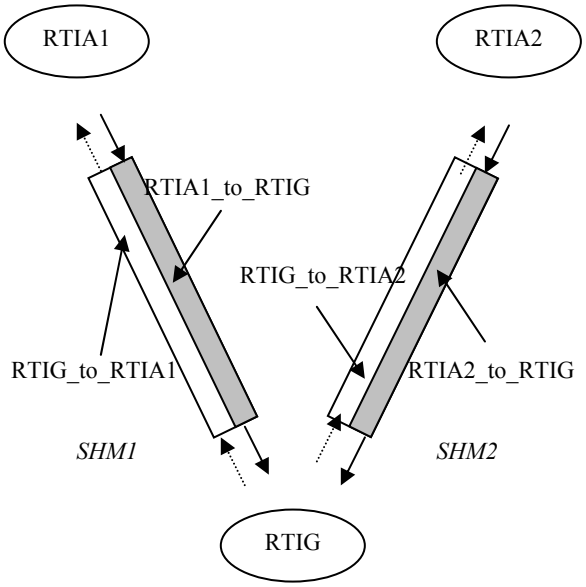


Figure 3: Shared memory segments and buffers

At time $T_1$, RTIA1 invokes an UAV request to the RTIG through the RTIA1_to_RTIG shared memory buffer. The *status* flag of the buffer is set to *full* and the RTIA1 process waits then for an acknowledge message from RTIG. At time $T_2$, the RTIA2 invokes accordingly an UAV request to the RTIA2_to_RTIG shared memory buffer. In the same way, RTIA2 will wait for an acknowledge message from the RTIG. At this time, the *status* flag of both RTIG_to_RTIA1 and RTIG_to_RTIA2, remains set to *empty*. Conversely, the status flag of both RTIA1_to_RTIG and RTIA2_to_RTIG have been set to *full*.

The RTIG process finds then the RTIA1_to_RTIG *status* flag full, and reads the incoming message. It sets the flag to *empty*, but the RTIA1 is waiting for an acknowledge message from the RTIG. It process accordingly the message and finally invokes the RAV service by writing the corresponding message into the RTIG_to_RTIA2 memory buffer.

At time $T_3$, the RTIA2 process finds an UAV message in the RTIG_to_RTIA2 memory buffer and inserts it within a list of incoming message. It waits then for an acknowledge message of its RAV request from the RTIG.

At time $T_4$, the RTIA1 process receives the acknowledge message from the RTIG in the RTIG_to_RTIA1 memory buffer, returning then to its main loop. In the same way, the RTIG process discovers an incoming message in the RTIA2_to_RTIG memory buffer, corresponding to the UAV service invocation of the RTIA2.

It process then the message accordingly and finally writes the responding message RAV into the RTIG_to_RTIA1 memory buffer, at time $T_5$. Finally, the RTIG sends the acknowledge message of the UAV service previously invoked by the RTIA2. At time $T_6$, the acknowledge message is received by the RTIA2 trough the RTIG_to_RTIA2 buffer. The RTIA2 returns then to its own main loop and delivers the stored messages to the federate when a *tick()* service is invoked.

## 5. Deploying CERTI over Kerrighed Operating System

As explained in section 3.2 Kerrighed SSI is attractive to achieve high availability of parallel or distributed applications, since it has to be considered as a distributed operating system dedicated to clusters, that provides process migration mechanisms to support load balancing. The SSI supports either the message passing programming model or the shared memory model. Notice that Kerrighed is able to dynamically migrate either processes or threads from a given cluster node to another target node. In our context of work, the distributed application is the CERTI Run-time infrastructure, built around RTIG, RTIAs and federates processes..

.The process migration mechanism is implemented as an internal operating system mechanism, fully transparent to users. This mechanism involves few Linux kernel modifications. More precisely, the process migration mechanism is based on the Linux kernel function **do_fork** allowing to restart the process on the destination node. A detailed technical description of the design of Kerrighed process migration is beyond the scope of this paper. Additional information can be found elsewhere [13].

Deploying a given application under the Kerrighed SSI operating system is straightforward. From a end user point of view, once the SSI is installed, two main directives are available. The *krgreboot* directive loads the Kerrighed SSI as well as the modified kernel

modules. Conversely, the *krgremove* directive unloads the SSI operating system. While CERTI is running on a cluster over the Kerrighed control, the SSI is able to share memory among corresponding processes and threads. Moreover, it optimizes resource utilization by implementing load balancing. It is important to understand that the Kerrighed-CERTI package, part of the HP-CERTI initiative, aims at achieving high availability of resources rather than high performances.

## 6. Preliminary Performance Results

In this section we discuss some preliminary performance results related to SHM-CERTI implementation of HLA. Performances provided by Kerrighed SSI operating system are not yet available. Furthermore, performances increase of CERTI deployed over Kerrighed are mainly expected from the ability of the SSI to dynamically migrate processes among available processor of a cluster, in order to optimize load balancing. The experimentation platform is made up of a Dell PowerEdge 6600 machine with four Xeon processors, 1.4 GHertz speed each and 8 Gbyte of shared memory, under Linux 2.4.18 kernel. Preliminary performance results have been obtained from the DMSO benchmarks running under the socket based CERTI and the SHM-CERTI.

### 6.1. Latency benchmark

This benchmark program measures RTI performance in terms of the elapsed time it takes to send and receive an attribute update. The latency is measured as a two-way latency by having the reflecting federates return the initial objects timestamp. The one-way latency is computed then by dividing the elapsed time by two. Comparative results have been obtained by running two federates and changing the updated/reflected attribute size from 128 bytes to 2048 bytes. Figure 4 shows two comparative results including mean latency and standard deviation, when 10 000 updates/reflects are performed (-u option).

| | **Socket CERTI** | | **SHM-CERTI** | |
|---|---|---|---|---|
| | Mean (ms) | SD | Mean (ms) | SD |
| **128** | 0.574 | 0.775 | 0.351 | 0.496 |
| **512** | 0.591 | 0.286 | 0.339 | 0.335 |
| **1024** | 0.577 | 0.365 | 0.456 | 0.478 |
| **2048** | 0.709 | 0.405 | 0.682 | 0.463 |

Figure 4: CERTI latency benchmark

These latency results can be compared with those obtained with the DMSO RTI-NG V6 under the same software/hardware configuration (figure 5)

| | **RTI-NG V6** | |
|---|---|---|
| | Mean (ms) | SD |
| **128** | 0.541 | 0.031 |
| **512** | 0.551 | 0.070 |
| **1024** | 0.589 | 0.051 |
| **2048** | 0.681 | 0.108 |

Figure 5: RTI-NG latency benchmark

### 6.2. Timadvance benchmark

This benchmark program measures RTI performance in terms of the rate at which time advance requests are processed. Results have been obtained with two federates, 20 timestep cycles (-c option) and 10 000 TimaAdvanceRequest per cycle. Figure 6 shows the mean grants/sec and the associated standard deviation, when lookahead varies from 0.1 to 10. These timeadvance benchmark results can be compared with those obtained with the DMSO RTI-NG V6 under the same software/hardware configuration (figure 7)

| | **Socket CERTI** | | **SHM-CERTI** | |
|---|---|---|---|---|
| | Mean | SD | Mean | SD |
| **0.1** | 4494.45 | 541.262 | 5110.32 | 927.939 |
| **1** | 3356.11 | 497.207 | 4303.63 | 1099.12 |
| **10** | 3917.24 | 367.424 | 6408.9 | 234.555 |

Figure 6: CERTI timeadvance benchmark

| | **RTI-NG V6** | |
|---|---|---|
| | Mean | SD |
| **0.1** | 756.457 | 131.858 |
| **1** | 987.606 | 188.879 |
| **10** | 792.233 | 205.943 |

Figure 7: RTI-NG timeadvance benchmark

### 6.3. Throughput benchmark

This benchmark program measures RTI performance in terms of the elapsed time it takes to perform a specified number of attribute update service call. The number of updates received is also tracked to determine whether the RTI drops any updates. Figure 8 gives the experimental results obtained for two participating federates, 20 cycles (-c option) and 10000 updates per cycle (-u option), when the attribute size varies from 128 bytes to 2048 bytes. Results include mean UAVs/sec and standard deviation.

| | **Socket CERTI** | | **SHM-CERTI** | |
|---|---|---|---|---|
| | Mean | SD | Mean | SD |
| **128** | 3828.55 | 307.13 | 4421.802 | 204.66 |
| **512** | 3679.741 | 239.36 | 4116.422 | 176.45 |
| **1024** | 3279.397 | 246.11 | 3678.043 | 41.56 |
| **2048** | 2526.201 | 142.08 | 4455.867 | 413.74 |

Figure 8: CERTI throughput benchmark

Finally, figure 9 shows the throughput benchmark results obtained with the RTI-NG V6.

| | **RTI-NG V6** | |
|---|---|---|
| | Mean | SD |
| **128** | 4916.593 | 139.56 |
| **512** | 3902.945 | 76.21 |
| **1024** | 3347.505 | 772.87 |
| **2048** | 2005.916 | 3.01 |

Figure 9: RTI-NG throughput benchmark

## 7. Conclusion and further steps

The future of HLA, as a widely adopted standard, depends on its ability to evolve in order to meet the current expectations of a growing simulation community. Motivations of the HP-CERTI initiative, is to actively participate in this effort, according to two main directions. First of all, making available HP-CERTI as a open-source RTI to the simulation community allows increased responsiveness to user needs and reduces cost of entry. Secondly, HP-CERTI allows facing the execution performance requirements of real-time decision support systems based on M&S. Therefore, HLA becomes a promising candidate for both statically and dynamically composing complex simulations within integrated platforms.

In this paper, we have discussed the two main components of the HP-CERTI package. The first one is based on an optimization of the communication protocol between RTIAs and RTIG through extensive use of shared memory. The second one offers higher availability of execution resources when the RTI is deployed under the control of the Kerrighed SSI operating system.

Future work includes both optimization and generalization. An obvious optimization would be to replace the socket based communications between libRTI and RTIA, by shared memory based exchanges. Nevertheless, preliminary performance measurements showed that message exchanges between the federate and its RTIA are not a bottleneck. An interesting generalization would be to keep both communication supports, TCP sockets and shared memory, within the RTI architecture. Such an HLA implementation would allow constructing large scale and fully distributed simulations, based on components (clusters of federates) sharing the same execution platform.

## 8. Acknowledgements

This work is carried out in the context of the COCA (High Performance Distributed Simulation and Models Reuse) Project, sponsored by the DGA/STTC (Délégation Générale pour l'Armement/Service des Stratégies Techniques et des Technologies Communes) of the French Ministry of Defense. Our partners in the COCA project, Capgemini company and IRISA are also kindly acknowledged.

## 9. References

[1] P. Siron, “Design and Implementation of a HLA RTI Prototype at ONERA”, Fall Simulation Interoperability Workshop, Orlando, September 13-18, 1998

[2] B. Breholee, P. Siron “CERTI: Evolutions of the ONERA RTI Prototype”, Fall Simulation Interoperability Workshop, Orlando, September 8-13, 2002

[3] D. Stratton, J. Miller, S. Parr, "Developing an Open Source RTI Community", Spring Simulation Interoperability Workshop, Arlington (VA) , April 18 – 22, 2004

[4] A. Tolk, "Composable Mission Spaces and M&S Repositories – Applicability of Open Standards", Spring Simulation Interoperability Workshop, Arlington (VA), April 18 – 22, 2004

[5] M. Adelantado, "Modeling and Simulation of Air Trafic Management Systems to Support the Decision Making Process: Present and Future Issues", SCS Advanced Simulation Technologies Conference, Business and Industry Symposium, Arlington (VA), April 18 – 22, 2004

[6] M. Adelantado, J. Latour, A. Vincent, P. Bonnet, "Distributed Simulation for Acquisition and Analysis of Future Operational Concepts", AIAA Journal of Aerospace Computing, Information and Communication, January 2004

[7] M. Adelantado, "Rapid Prototyping of Airport Advanced Operational Systems and Procedures through Distributed Simulation", SCS Journal on Transactions of the Society for Modeling and Simulation International, Special Issue on Simulation of Air Traffic, January 2004

[8] C. Morin, P. Gallard, R. Lottiaux, G. Vallée, "Towards an Efficient Single System Image Cluster Operating System", in Future Generation Computer Systems, 20(2), January 2004

[9] P. Gallard, C. Morin, "Dynamic Streams for Efficient Communications between Migrating Processes in a Cluster", in Parallel processing Letters, 13(4), December 2003

[10] G. Vallée, R. Lottiaux, L. Rolling, J-Y. Berthou, I. Dutka-Malhen, C. Morin, "A Case for Single System Image Cluster Operating Systems: Kerrighed Approach", in Parallel Processing Letters, 13(2), June 2003

[11] D. Margery, G. Vallée, R. Lottiaux, C. Morin, J-Y. Berthou, "Kerrighed: a SSI Cluster OS Running OpenMP", in Proc. 5th European Workshop on OpenMP, September 2003

[12] M. D. Petty, E. W. Weisel, "A Composability lexicon", Proceedings of the Spring Interoperability Workshop, Orlando (FL), March 30 – April 4, 2003

[13] G. Vallée, C. Morin, J-Y. Berthou, I. Dutka Malen, R. Lottiaux, "Efficient Process Migration based on Gobelins Distributed Shared Memory", Internal Report n° 4518, INRIA, July 2002

## Author Biography

**MARTIN ADELANTADO** was graduated from a French High School for Engineers in Computer Science (ENSEEIHT) in 1979, and received his doctorate in 1981. He is an ONERA (French Aeronautics and Space Research Agency) Senior Scientist and works at the Aeronautics Systems Research Unit of the Long Term Design and Systems Integration Department (DPRS/SAE). He received his "Habilitation à diriger des Recherches" French diploma in May 2004. His fields of interest include simulation, real-time systems, distributed systems and M&S of complex systems. He is currently in charge of the design and the development of a set of modeling and simulation tools for fast prototyping of both air and ground traffic systems.

**PIERRE SIRON** was graduated from a French High School for Engineers in Computer Science (ENSEEIHT) in 1980, and received his doctorate in 1984. He is currently a Research Engineer at ONERA and he works in parallel and distributed systems and computer security. He is a member of the Design and Validation of Computer Systems research unit. He received his "Habilitation à diriger des Recherches" French diploma in January 2003.

**MARC BETOULE** is currently student at ENSAE, a French High School for Engineers in Aerospace Sciences (Ecole Nationale Supérieure de l'Aéronautique et de l'Espace), Toulouse.